BRL-MR-3784

4

MEMORANDUM REPORT BRL-MR-3784

BRL

AD-A213 270

PROGRAM TRANSFORMATION WITH ABSTRACT RELATION ALGEBRAS

PAUL BROOME

OCTOBER 1989

U.S. ARMY LABORATORY COMMAND

BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

89 10 12 024

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE	Form Approved OMB No 0704-0188
1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Distribution Unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) BRL-MR- 3784	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION System Engineering & Concepts Analysis Division; 6b OFFICE SYMBOL (If applicable) SLCBR-SE-C	7a NAME OF MONITORING ORGANIZATION US Army Ballistic Research Laboratory
6c ADDRESS (City, State, and ZIP Code) USA Ballistic Research Laboratory Aberdeen Proving Ground, MD 21005-5066	7b ADDRESS (City, State, and ZIP Code) ATTN: SLCBR-D Aberdeen Proving Ground, MD 21005-5066
8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

PROGRAM TRANSFORMATION WITH ABSTRACT RELATION ALGEBRAS

12 PERSONAL AUTHOR(S)

PAUL H. BROOME

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Memorandum	FROM ______ TO ______		

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Program Transformation; Relation Algebra; Logic Programming; Functional Programming Organizational Reasoning

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Our objective is to satisfy two seemingly opposing constraints on program design. We want to begin a program as an obvious formal specification but also insist upon efficient execution.

Our approach is to find expressive, model independent program constructs with algebraic properties that can show logical equivalence between a clear specification and an efficient program.

Our solution is an implementation of some of the relations, relation operators, and equations that comprise Tarski's Q-relation algebra. The key to this solution is a single operator for linear recursions that satisfies two equations. The first merges recursions and the second propagates constraints for early pruning of the nonproductive branches of the computation tree.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a NAME OF RESPONSIBLE INDIVIDUAL PAUL H. BROOME	22b TELEPHONE (Include Area Code) 301-278-6883; 22c OFFICE SYMBOL SLCBR-SE-C

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Contents

1 Introduction

One of the most attractive program development methods is transformational programming. In this report we explore transformation oriented language design. We start with the assumptions that

1. Programs are to be initially written as straightforward, understandable, executable specifications. More efficient, but probably less readable, programs are derived through correctness preserving transformations.
2. Functional and relational operators provide an appropriate context for specifications. They have the algebraic properties required for the transformations.

The central contribution is the application of Tarski's RA and Q-relation algebras [1] to program transformation. We find the notation is a convenient combination of logic programming and functional programming. This middle ground has the express-ibility of the former and the manipulability of the latter because it provides for higher order operations in a first order way. A key contribution is a single operator for linear recursions over regularly structured objects along with equations for merging loops and propagating constraints.

2 Motivation

Operators have been largely ignored in logic programming languages. In general the greater the amount of pattern directed invocation of procedures there is in a language, the less the dependence on operators. This need not and should not be the case as both concepts are an important part of modern programming languages, and operators help reason about programs at a high level.

An operator in Prolog is a program clause, describing an n-ary relation, with a variable that is called as a predicate in the body of the clause. Warren [2] described how to define higher order operators in Prolog and argued that extending the Prolog language definition to include them was, in large part, unnecessary. Indeed, their definitions are simple but they are awkward to use.

For example, the following Prolog program defines a 4 argument relation *iterate* and exhibits the difficulties with higher order operators in Prolog. The first argument of *iterate* is a sequence and the second, F, is a predicate of 3 arguments. The third argument, Z, is normally a right identity for F although this is not required. It is the answer supplied when iterated over an empty list. Finally, the last argument for *iterate* is the result.

EXAMPLE 1 *A Prolog iterate operator*

$$
\begin{array}{l}
iterate([\,], F, Z, Z).\\
iterate([X|L], F, Z, Y) :-\\
\quad iterate(L, F, Z, Y_0),\\
\quad F(X, Y_0, Y).
\end{array}
$$

Operators defined this way are not often directly applicable as there is no convention about the number of arguments in a predicate. In particular, *iterate* expects a function, such as *plus*, with exactly three arguments such that the first two are different 'inputs' and the third is an answer. Secondly, lists are 'wired into' the first argument. For example, it cannot be used to generate the first N integers such as with APL's *iota*. Finally, this operator does not describe all iterations over lists. An example is *map* which applies the same function in a one-to-one fashion to each item of a list. Each of these linear recursions requires a new definition and a combinatorial increase in the effort to reason about their combinations.

If a program construct is to be manipulated then it must satisfy some algebraic properties. These properties are most applicable if the construct is abstractly defined, independently of a model. The following preliminaries are model independent.

3 Preliminaries

We will interpret the notation as both mathematical symbols and as program constructs but recognize that the two are distinct. Within definitions and arguments for correctness we follow Tarski's extended predicate logic $\mathcal{L}^+$ [1]. Propositions are statements of the algebraic properties of relation operators. However, definitions are also interpreted as program clauses and propositions as properties of program forming operations. We view every program as a relation between inputs and outputs with the proviso that the viewpoint can be reversed, when feasible, to construct inputs from outputs. These transformations reduce the size of the computation tree and are applicable to both sequential and OR-parallel computations if conjuctions of subgoals are solved from left to right.

The algebraic terminology categorizes a portion of the large collection of relation equations. It also helps describe algorithms abstractly, independently of a model. The most specific structures are relation algebras and Q-relation algebras as described in [1]. Table 1 is a summary of notation.

Symbol	Meaning
$A - Z$	variables
$a - z$	constants or function symbols
$\mathcal{T}$	set of terms
$\mathcal{S}$	set of ground terms
$\mathcal{B}(\mathcal{S})$	set of binary relations on $\mathcal{S}$
$\wedge$	logical and
$\vee$	logical or
$\neg$	logical negation
$\exists$	existential quantifier
$\forall$	universal quantifier
$\leftrightarrow$	if and only if
$\Leftarrow$	is provable if
$\equiv$	defined or proved equivalent to
$[\_\|\_]$	ordered couple

Table 1: Summary of notation

The theory of relations is one of the most developed branches of logic[1,3,4]. In particular, the calculus of binary relations follows the well understood laws of Boolean algebra.

A Boolean algebra is a structure $\langle \mathcal{U}, \cup, \overline{} \rangle$. Although there are no implicit assumptions about the underlying universe $\mathcal{U}$, we are most interested in algebras defined on $\mathcal{B}(\mathcal{S})$. The following is an sample axiomatization.

$$(X \cup Y) \cup Z \equiv X \cup (Y \cup Z), \tag{1}$$
$$X \cup Y \equiv Y \cup X, \tag{2}$$
$$X \equiv \overline{(\overline{X} \cup Y)} \cup \overline{\overline{X} \cup \overline{Y}}. \tag{3}$$

A relation algebra is a structure $\langle \mathcal{U}, \cup, \overline{}, \circ, \breve{}, id \rangle$. The common language describes these operations as union, complementation, composition, converse, and the identity. A summary of the relation operators appears in Table 2. The following is an axiomatization for RA [1,5].

$$(F \cup G) \cup H \equiv F \cup (G \cup H), \tag{4}$$
$$F \cup G \equiv G \cup F, \tag{5}$$
$$F \equiv \overline{(\overline{F} \cup G)} \cup \overline{(\overline{F} \cup \overline{G})}, \tag{6}$$

Relation Symbol	Meaning
ϕ	empty relation
ω	universal relation
$R \cup S$	union of relations
$R \cap S$	intersection of relations
$R \square S$	ordered coupler of relations
$R \circ S$	relative product or composition
$\overline{R}$	complement of the relation R
$R^{\smile}$	converse of a relation
R^{*}	monoid closure of R
$[R \mid S]$	relational constructor of couples
$\pi(D, S, C)$	linear recursion
id	identity relation
di	diversity relation
hd	first conjugated projection
tl	second conjugated projection

Table 2: Summary of Relations and Relation Operators

$$F \circ (G \circ H) \equiv (F \circ G) \circ H, \tag{7}$$
$$(F \cup G) \circ H \equiv F \circ H \cup G \circ H, \tag{8}$$
$$F \circ id \equiv F, \tag{9}$$
$$F^{\smile\smile} \equiv F, \tag{10}$$
$$(F \cup G)^{\smile} \equiv F^{\smile} \cup G^{\smile}, \tag{11}$$
$$(F \circ G)^{\smile} \equiv G^{\smile} \circ F^{\smile}, \tag{12}$$
$$F^{\smile} \circ \overline{F \circ G} \cup \overline{G} \equiv \overline{G}. \tag{13}$$

From these fundamental operations we can abstractly define other relations and relation operators. Positive, logical definitions of intersection $F \cap G$ and the universal relation ω are preferred. They are more easily implemented and constructive proofs are more intuitive. Implementation considerations for negations such as complement and diversity di are discussed in the next section.

$$F \cap G \equiv \overline{\overline{F} \cup \overline{G}}, \tag{14}$$
$$\omega \equiv id \cup \overline{id}, \tag{15}$$
$$\phi \equiv \overline{\omega}, \tag{16}$$

$$di \equiv \overline{id}. \tag{17}$$

It is easy, but tedious, to check that the following logical definitions for these relation operators satisfy the properties of an RA.

$$\forall XY\{X\ (F \cap G)\ Y \leftrightarrow X\ F\ Y \wedge X\ G\ Y\}, \tag{18}$$
$$\forall XY\{X\ (F \cup G)\ Y \leftrightarrow X\ F\ Y \vee X\ G\ Y\}, \tag{19}$$
$$\forall XY\{X\ (F \circ G)\ Y \leftrightarrow \exists\{ZX\ F\ Z \wedge Z\ G\ Y\}\}, \tag{20}$$
$$\forall XY\{X\ F^{\smile}\ Y \leftrightarrow Y\ F\ X\}, \tag{21}$$
$$\forall X\{X\ id\ X\}, \tag{22}$$
$$\forall XY\{X\ \omega\ Y\}, \tag{23}$$
$$\forall XY\{X\ \overline{F}\ Y \leftrightarrow \neg(X\ F\ Y\}, \tag{24}$$
$$\forall XY\{X\ di\ Y \leftrightarrow \neg(X\ id\ Y\}. \tag{25}$$

Other simple properties of union and intersection follow.

$$A \cup \phi \equiv A, \tag{26}$$
$$\phi \cup A \equiv A, \tag{27}$$
$$(A \cap B) \cap C \equiv A \cap (B \cap C), \tag{28}$$
$$A \cap \omega \equiv A, \tag{29}$$
$$\omega \cap A \equiv A. \tag{30}$$

For any given $RA = \langle \mathcal{U}, \cup, \cap, \overline{\ }, \phi, \omega \rangle$, two elements a, b are called ***conjugated quasiprojections*** if $a^{\smile} \circ a \subseteq id$, $b^{\smile} \circ b \subseteq id$, and $a^{\smile} \circ b \equiv \omega$. An RA is called a ***Q-relation algebra*** if its universe contains some conjugated quasiprojections.

These definitions say nothing about the intended realization, and their abstractness is what makes them appropriate for describing broadly applicable operations. In [1] the intended interpretation is encapsulated into a membership relation **E**. All laws of RA and a Q-relation algebra hold for any definition of **E**, and **E** connects the abstract relation algebras with a particular universe over which the relations are to range.

In particular, a Q-relation algebra defined over a nontrivial universe contains ordered pairs or couples and the conjugated quasiprojections suggest that there are tools for selecting components of these couples. We define two such selectors over ordered couples called ***conjugated projections*** and represent them by ***hd*** and ***tl***.

$$\forall XY\{[X|Y]\ hd\ X\}, \tag{31}$$
$$\forall XY\{[X|Y]\ tl\ Y\}. \tag{32}$$

Note that they satisfy $hd^{\smile} \circ hd \equiv id, tl^{\smile} \circ tl \equiv id$, and $hd^{\smile} \circ tl \equiv \omega$. Thus they qualify as conjugated quasiprojections. These projections select components of ordered couples. The following operators construct or perform related functions on couples.

$$[F|G] \equiv (F \circ hd^{\smile}) \cap (G \circ tl^{\smile}). \tag{33}$$
$$F \Box G \equiv [hd \circ F | tl \circ G]. \tag{34}$$

The following properties hold for construction and coupling operators. They all have similar proofs

$$[F \cup G|H] \equiv [F|H] \cup [G|H]. \tag{35}$$
$$[F|G \cup H] \equiv [F|G] \cup [F|H]. \tag{36}$$
$$[F|G] \cap [H|K] \equiv [(F \cap H)|(G \cap K)]. \tag{37}$$
$$(F \Box G) \circ (H \Box K) \equiv (F \circ H) \Box (G \circ K). \tag{38}$$
$$(F \Box G \circ W) \circ (R \Box S) \equiv (F \Box G) \circ (R \Box W \circ S). \tag{39}$$
$$(F \circ W \Box G) \circ (R \Box S) \equiv (F \Box G) \circ (W \circ R \Box S). \tag{40}$$
$$(F \Box G)^{\smile} \equiv (F^{\smile} \Box G^{\smile}). \tag{41}$$
$$(F \cup G) \Box H \equiv (F \Box H \cup G \Box H). \tag{42}$$
$$F \Box (G \cup H) \equiv (F \Box G \cup F \Box H). \tag{43}$$
$$(F \Box G) \cap (H \Box K) \equiv (F \cap H) \Box (G \cap K). \tag{44}$$

4 Application to programs

The logical definitions of $\cup, \cap, \circ, ^{\smile}, hd, tl, \Box, [\,|\,], id$, and ω (18-23) suggest an obvious implementation with an SLD-resolution logic programming system in the logic of predicates of three variables. For example, the goal $X \; R \; Y$ would be represented as $p(X, R, Y)$. Diversity and complement are discussed in section 4.1.

Note that no sentence introduces variables into the relation argument. Thus every literal in the body of a program clause of the form $p(X, R, Y)$ will contain only variables R that were named in the clause head. Therefore if a goal is $p(X, R, Y)$ where R is ground, then every relation argument in every subgoal will be ground. We require that R be selected from a finite number of possible relations and relational constructs.

We shall read $R \equiv S$ as 'R is equivalent to, but should be rewritten as, S.' But this is represented in a logic program as the clause $p(X, R, Y) : -p(X, S, Y)$. Conjunctions are solved from left to right. Our intent is to match control with the order in which variables are bound. Therefore we partition the arguments of a relation into two structured components, an 'input' and an 'output.' Predicates of single arguments are extended to two arguments, both of which are the same value.

The FP systems of Backus [6] also depend upon operators, particularly composition, to form more complex programs out of simpler ones. Berghammer and Ziever [7] have given a relation algebra semantics for FP-like languages. Mili, Desharnais, and Mili [8] have given heuristics for the design of deterministic programs from relational equations. This paper describes a transformation system based on the operators, relations, and equations of a Q-relation algebra.

4.1 Diversity and Complement

The identity $X\ id\ Y$ requires unification of terms X and Y. The diversity relation depends on antiunification [9] and is denoted $A\ di\ B$. If A and B are both ground constants then $A\ di\ B$ will fail or succeed depending on whether they are the same or different constants. On the other hand, if they contain unbound variables then the environment is extended to include that information as inequality constraints.

Terms with structure are solved recursively in a manner similar to unification. If the two terms A and B have either different principal functors or different numbers of arguments then antiunification succeeds without new inequality constraints. On the other hand, with the same principal functor and, say N, arguments, diversity in any argument is enough for antiunification to succeed. Thus possibly N new choice points are created by recursing the algorithm on these arguments. If one of the corresponding pairs of arguments can be determined to be different then no other alternatives need be considered as the algorithm terminates successfully.

For example, consider the problem $f(X, g(a))\ di\ f(u, g(Y))$. The two solutions (inequality constraints) are $X \neq u$ and $Y \neq a$. On the other hand, the subgoal $f(X, a)\ di\ f(Y, b)$ succeeds with no new inequality constraints.

Negation as finite failure cannot compute a complement. The goal $\exists XY, X\ \overline{R}\ Y$ is equivalent to $\exists XY, \neg X\ R\ Y$ but negation by failure instead determines $\neg(\exists XY, X\ R\ Y)$. Although negation as finite failure has the usual logical interpretation for ground goals [10], in the face of transformation we cannot be assured that a variable will be ground.

Constructive negation is an alternative to negation by failure. The elements of $\overline{R}$ are not actually constructed, but instead the resolution procedure is extended with inequality constraints. Thus, solutions to nonground, negative subgoals are constructed as a set of inequality constraints. Constructive negation has both a clean semantics and the advantage of speeding up some computations [11].

Although relation complement is expressive, it is not monotonic because $R \subseteq S$ does not imply $\overline{R} \subseteq \overline{S}$. Transformations on programs built with $\overline{R}$ may become invalid if R is extended. We will depend most heavily upon the other relation operators including the diversity relation and avoid relation complement when possible.

4.2 Sequence construction and selection

A convenient way of sharing an input to more than one function is with Backus' constructor functional. We defined a similar constructor operation on relations. The ordered coupling operator creates an ordered couple as an output from a pair of inputs in a one to one fashion. Both operations are strict. While complex structures are made with constructors, they are taken apart with *hd* and *tl*. These are called *selectors* by Backus[6]. Selectors disassemble what the constructors and ordered couplers build.

While success of $X\ hd \circ F\ Y$ indicates X is a couple, $X\ \overline{hd} \circ G\ Y$ would not fail immediately as $X\ \overline{hd}\ Z$ would be delayed until Z were bound. Thus we need a model of lists with a special symbol such as [ ] to indicate the end of list. A sequence of one element A will be represented as $[A]$, that is $[A|B]$ where $B = [\,]$. The function *null* tests sequences for emptiness and is an identity on [ ]. That is, *null* is just the single pair [ ].[ ].

We also define a relation to deposit this symbol into a list construction. This empty sequencer is a constant function that ignores domain elements, returning the object [ ]. We also represent this function with the symbol [ ]. It is defined as $[\,] \equiv \smile \circ null$. The following are some simple properties of *null* and [ ].

$$[A|B] \circ null \equiv \phi, \tag{45}$$
$$(A \Box B) \circ null \equiv \phi, \tag{46}$$
$$null \circ (A \Box B) \equiv \phi, \tag{47}$$
$$[\,] \circ null \equiv [\,], \tag{48}$$
$$null \circ [A|B] \equiv [null \circ A | null \circ B]. \tag{49}$$

5 Linear recursion

The *closure* of a relation R is R repeatedly composed with itself and is defined as $R^* \equiv id \cup R \circ R^*$. Consider the program that finds the greatest common divisor of two numbers. Here the realization is the natural numbers and *subtract* is defined only on that set. Thus $[3|5]$ *subtract* Y would fail but $[5|3]$ *subtract* 2 succeeds.

EXAMPLE 2 *Greatest common divisor*

$$gcd \equiv ((id \cup [tl|hd]) \circ [subtract|tl])^{*} \circ (hd \cap tl).$$

This simple program starts with two numbers and continually subtracts the second number from the first until the numbers are the same. If necessary, it reorders the numbers so that the largest is first. A trace of the computation on the couple $[6|18]$ is $[6|18] \Rightarrow [18|6] \Rightarrow [12|6] \Rightarrow [6|6] \Rightarrow 6$. The monoid closure is not expressive enough and is naturally oriented to a universe of elements without structure. We define a linear recursion operator that extends the effect of the ordered coupler to lists and list-like structures. This operator divides the structure into a couple with D, solves for base cases with S, then combines the elements of a couple with C.

$$\pi(D, S, C) \equiv S \cup D \circ (id \square \pi(D, S, C)) \circ C. \tag{50}$$

We can define monoid closure with π as $R^{*} \equiv \pi([\omega|R], id, tl)$. Also, we can define *map* to apply the effect of a relation to each item of a sequence. Thus for example, the goal $[2, 3]$ *map*$(id \cup sub1)$ Y has four solutions for Y. The solutions would be $Y = \{[2,3], [2,2], [1,3], [1,2]\}$. The definition is

$$map(R) \equiv \pi(id, null, R \square id). \tag{51}$$

In our relations we collect inputs into a single structured input. For example, the Prolog goal *append*(A, B, C) is written as $[A|B]$ *append* C. On the other hand, predicates of a single argument become two argument, subrelations of the identity. An example is 3 *odd* 3. These extensions are important because they allow us to use higher order operators with Boolean valued relations to define new predicate operators.

All recursions that follow will be given in terms of π. They are preliminary to the example transformations. The next definition is the function that cumulatively concatenates a sequence of sequences.

$$conc \equiv \pi(id, null, append), \tag{52}$$

$$append \equiv \pi([hd \circ hd|tl \square id], (null \square id) \circ tl, id). \tag{53}$$

We can couple an item to each element of a sequence with either *distr* or *distl* as in [6]. These functions can be simply defined in terms of π and the definitions expose their inherent symmetry.

$$distr \equiv \pi([hd \square id|tl \square id], hd \circ null, id), \tag{54}$$

$$distl \equiv \pi([id \square hd|id \square tl], tl \circ null, id). \tag{55}$$

To see what is happening here consider, for example, the function *distr*. It inputs an ordered couple, a sequence and an item. The result is a sequence of couples each of which has the given item as a second component. Schematically this is

$$[[A_1,\cdots,A_N]|C]\ \mathit{distr}\ [[A_1|C],[A_2|C],\cdots,[A_N|C]].$$

Many relations can now be concisely defined in a single expression. For example, we can define *member* on a couple, an element and a sequence. Thus *member* is simply a predicate (an identity) that tests the element for membership in the sequence. For example both ($[3|[2,3,4]]$ *member* $[3|[2,3,4]]$) and ($[3|[5]]$ *nonmember* $[3|[5]]$) are true.

$$\mathit{select} \equiv \pi(id, hd, tl). \tag{56}$$

$$\mathit{member} \equiv [hd \cap (tl \circ \mathit{select})|tl]. \tag{57}$$

$$\mathit{nonmember} \equiv [hd|distl \circ map((hd \circ di) \cap tl)]. \tag{58}$$

The following equations characterize not only some forms of loop merging but can also propagate constraints. Since control is left to right, constraints on search are best performed as soon as possible. The following equation enables further propagation of constraints.

PROPOSITION 1 Merging linear recursions.

$$\pi(D, S \circ null, id) \circ \pi(id, null \circ T, C) \equiv \pi(D, S \circ null \circ T, C).$$

The proof is by induction over relations defined on a well founded set. We unfo each π-term in the expression $\pi(D, S \circ null, id) \circ \pi(id, null \circ T, C)$ to arrive at

$$(S \circ null \cup D \circ (id \Box \pi(D, S \circ null, id))) \circ (null \circ T \cup (id \Box \pi(id, null \circ T, C)) \circ C).$$

Equations 46 and 47 eliminate the cross terms to give

$$S \circ null \circ null \circ T \cup D \circ (id \Box \pi(D, S \circ null, id)) \circ ((id \Box \pi(id, null \circ T, C)) \circ C).$$

Now we can apply 38 to bring the two π-terms together.

$$S \circ null \circ T \cup D \circ (id \Box \pi(D, S \circ null, id) \circ \pi(id, null \circ T, C)) \circ C.$$

Applying the induction hypothesis we now have

$$S \circ null \circ T \cup D \circ (id \Box \pi(D, S \circ null \circ T, C)) \circ C.$$

Folding, we have $\pi(D, S \circ null, id) \circ \pi(id, null \circ T, C) \equiv \pi(D, S \circ null \circ T, C)$.

PROPOSITION 2 Propagation of constraints.

$$\pi(D, S, (R \Box id) \circ C) \equiv \pi(D \circ (R \Box id), S, C).$$

Again D must map a well founded set into lists. By induction and equation 34

$$\begin{aligned} \pi(D, S, (R \Box id) \circ C) &\equiv S \cup D \circ (id \Box \pi(D, S, (R \Box id) \circ C)) \circ (R \Box id) \circ C, \\ &\equiv S \cup D \circ (R \Box id) \circ (id \Box \pi(D, S, (R \Box id) \circ C)) \circ C, \\ &\equiv S \cup D \circ (R \Box id) \circ (id \Box \pi(D \circ (R \Box id), S, C)) \circ C, \\ &\equiv \pi(D \circ (R \Box id), S, C). \end{aligned}$$

The relation R has changed positions. Now $R \Box id$ tests on the way down, before any large structure has been built, instead of on the way back up. A simple consequence is the equation $map(F) \circ map(G) \equiv map(F \circ G)$. This follows from these two equations and the fact that $map(R) \equiv \pi(id, null, R \Box id)$.

If a concatenation of sequences is required to be empty, we can avoid the concatenation by requiring that each subsequence of the sequence be empty.

$$conc \circ null \equiv map(null) \circ [\,]. \tag{59}$$

This rule follows by induction from the definitions of *conc*, *null*, and *map* with equations 8, 49, 39, and 40.

6 Transformations

We use the equations developed here to automatically accomplish a nontrivial program transformation that is interesting for two reasons. Primarily, it does not rely on the Burstall and Darlington fold/unfold technique [12] but is instead directed by the operator definitions and equations between relations. Therefore the method is easily mechanized. Secondly, unlike Hogger's techniques [13], it is carried out using relation level reasoning without resorting to object level variables. This saves symbols and makes the derivation more concise and broadly applicable.

The problem has two parts. The first part finds the list intersection of two lists. We may consider this program as a trivial example of a library program. Library programs may well be written efficiently but combinations are often inefficient. If two programs are written so that constraints are applied as early as possible, before alternatives are created, the composition may have some constraints that are applied too late.

A programmer should not expect to be penalized with the inefficiencies of library programs. Our second program exhibits this problem. It is a clear, easy to understand program that simply tests to see if two sequences are disjoint. Using the rules developed earlier, we remove the inefficiencies in that program.

6.1 Disjointedness as empty intersection

The first part of this program finds the intersections of two lists by distributing one list over the elements of the other, then finding those other elements that are members of the list.

$$list\_intersect \equiv distr \circ map(member \circ [hd] \cup nonmember \circ [\,]) \circ conc. \tag{60}$$

The program *disjoint* determines if two lists are disjoint by simply testing for an empty intersection. To return a positive answer we extend the model with the identity relation on the constant *yes*. As defined, this program is unnecessarily inefficient although it is understandable in terms of its parts.

$$disjoint \equiv list\_intersect \circ null \circ \omega \circ yes. \tag{61}$$

The transformation system applies the relation equations outwards in, from left to right. After each successful rewrite the enclosing expression is attempted before deeper optimization is done to the subexpressions[14]. The first two recursions are within the definition of *list_intersect*. Let *buildone* represent the expression $member \circ [hd] \cup nonmember \circ [\,]$. Then,

$$list\_intersect \circ null \equiv distr \circ map(buildone) \circ conc \circ null.$$

We direct our attention to the two recursions in $distr \circ map(buildone)$ Both *distr* and *map* are defined with π so we merge and simplify them.

$$\begin{aligned}
distr \circ map(buildone) \;&\equiv \pi([hd\Box id|tl\Box id], hd \circ null, id) \circ map(buildone),\\
&\equiv \pi([hd\Box id|tl\Box id], hd \circ null, id) \circ \pi(id, null, buildone\Box id),\\
&\equiv \pi([hd\Box id|tl\Box id], hd \circ null, buildone\Box id),\\
&\equiv \pi([hd\Box id|tl\Box id] \circ (buildone\Box id), hd \circ null, id),\\
&\equiv \pi([hd\Box id \circ buildone|tl\Box id \circ id], hd \circ null, id),\\
&\equiv \pi([hd\Box id \circ buildone|tl\Box id], hd \circ null, id).
\end{aligned}$$

Next, $conc \circ null$ is expanded to $map(null) \circ [\,]$ which is $\pi(id, null, null\Box id) \circ [\,]$. Thus $list\_intersect \circ null$ is now two recursions followed by $[\,]$. This is

$$\pi([hd\Box id \circ buildone|tl\Box id], hd \circ null, id) \circ \pi(id, null, null\Box id) \circ [\,].$$

Once again we merge recursions and propagate constraints to obtain

$$\begin{aligned}
list\_intersect \circ null \;&\equiv \pi([hd\Box id \circ buildone|tl\Box id], hd \circ null, null\Box id) \circ [\,],\\
&\equiv \pi([hd\Box id \circ buildone|tl\Box id] \circ null\Box id, hd \circ null, id) \circ [\,],\\
&\equiv \pi([hd\Box id \circ buildone \circ null|tl\Box id \circ id], hd \circ null, id) \circ [\,].
\end{aligned}$$

Now *buildone* is a union, over which we can distribute *null* to obtain $buildone \circ null \equiv member \circ [hd] \circ null \cup nonmember \circ [\,] \circ null$. In addition, one branch of the union goes away as $[hd] \circ null$ always fails. Therefore, we have

$$\begin{aligned}
disjoint \;&\equiv \pi([hd\Box id \circ (nonmember \circ [\,] \circ null)|tl\Box id \circ id], hd \circ null, id) \circ \omega \circ yes,\\
&\equiv \pi([hd\Box id \circ (nonmember \circ [\,])|tl\Box id], hd \circ null, id) \circ \omega \circ yes.
\end{aligned}$$

This is the order in which the implemented transformation system rewrites the original program. The mechanically derived program for *disjoint* tests for nonmembership before creating large structures. When the two lists differ at their first components, the remaining components need not be tested. For two lists of 30 elements, this program is approximately 100 times faster than the original.

This is a significant speedup, but we should note that an arbitrary amount of speedup is possible with the converse operator. For example with sequential, left-to-right AND, the cost of solving $X\ (parent^{*})^{\smile}\ Y$ is much greater than the cost of solving $X\ (parent^{\smile})^{*}\ Y$ for a large family tree [15].

6.2 Eliminating intermediate lists

Wadler describes a *deforestation* method for avoiding intermediate lists [16]. He applies it to a program to compute the sum of squares of numbers between 1 and N. The method uses the unfold-fold method on recursive equations and applies to deterministic programs. Transformations based on relation algebras extend these techniques to nondeterministic computations in a verifiable way.

The program performs just three main steps. The program constructs a sequence from N to 1, squares every number, and sums the sequence. The first interesting observation is that all of these operations are described by the π operator.

$$\begin{aligned}&\pi([id|sub1], eq0 \circ [\,], id)\circ\\ &\pi(id, null, sqr\Box id)\circ\\ &\pi(id, null \circ \omega \circ eq0, plus).\end{aligned}$$

The realization for a Q-relation algebra requires the extra relations $eq0$, $sub1$, sqr, and $plus$. These have the obvious definitions except that $eq0$ is just the single pair $\{0, 0\}$.

The first two recursions can be merged, with proposition 1, and the sqr operation can be brought forward, with proposition 2, to obtain

$$\pi([sqr|sub1], eq0 \circ [\,], id) \circ \pi(id, null \circ \omega \circ eq0, plus).$$

Once again the two recursions can be merged to obtain the result $\pi([sqr|sub1], eq0, plus)$.

7 Conclusion

We have learned that the Q-relation algebras form a firm foundation for a transformation oriented programming language. The abstract relation operators are appropriate for describing the generic constructs that often arise in programming.

We have defined a model independent operator π that describes linear recursions and have given two broadly applicable equations that merge recursions and propagate constraints. They provide an equational method for reasoning about nondeterministic computations.

The system based on these equations transforms these sample programs so that the result is essentially a different algorithm. This implementation shows that we can in some cases build new programs on previously constructed ones without the usual efficiency penalty from the combination.

Acknowledgements

These results would not have been possible without the valuable discussions I've had with Tom Myers and Dan Chester. Many thanks go to Roger Maddux for the time he has taken to explain relation algebras. Thanks also go to Rich Kaste for reviewing this work and to Brint Cooper, Jerry Andersen, and Jagdish Chandra for their support.

References

[1] A. Tarski & S. Givant, "A formalization of set theory without variables," in *Colloquium publications* #41, American Mathematical Society, Providence, Rhode Island, 1987.

[2] D.H.D. Warren, "Higher-order extensions to Prolog: are they needed?," in *Machine Intelligence 10*, Hayes, Michie and Pao, ed., John Wiley & Sons, New York, NY, 1982, 441–454 .

[3] C. S. Peirce, "On the Algebra of Logic: A Contribution to the Philosophy of Notation," *American Journal of Mathematics* VII (1885).

[4] A. Tarski, "On the calculus of relations," *The Journal of Symbolic Logic* 6 (1941), 73–89.

[5] Roger D. Maddux, "Introductory course on relation algebras, finite-dimensional cylindric algebras, and their interconnections," 1988, Notes.

[6] John Backus, "Can Programming Be Liberated from the von Neumann Style?; A Functional Style and Its Algebra of Programs," *Comm. ACM* 21 (1978), 613–641.

[7] R. Berghammer & H. Ziever, "Relational Algebraic Semantics of Deterministic and Nondeterministic Programs," *Theoretical Computer Science* 43 (1986), 123–147.

[8] A. Mili, J. Desharnais & F. Mili, "Relational Heuristics for the Design of Deterministic Programs," *Acta Informatica* 24 (1987), 239–276.

[9] A. Colmerauer, "Equations and inequations on finite and infinite trees," in *FGCS'84 Proceedings*, 1984, 85–99.

[10] Joxan Jaffar, Jean-Louis Lassez & John Lloyd, "Completeness of the Negation as Failure Rule," in *Proceedings of the 1983 IJCAI*, 1983, 500–506.

[11] D. Chan, "Constructive Negation Based On the Completed Database," in *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, R. A. Kowalski and K. A. Bowen, ed., MIT Press, Cambridge, MA, 1988, 111–125.

[12] R.M. Burstall & John Darlington, "A Transformation System for Developing Recursive Programs," *J. Assoc. Comput. Mach.* 24 (Jan., 1977), 44–67.

[13] C.J. Hogger, "Derivation of Logic Programs," *J. Assoc. Comput. Mach.* 28 (Apr., 1981), 372–392.

[14] Noor Islam, Tom Myers & Paul Broome, "A Simple Optimizer for FP-like Languages," in *Proc. ACM Conf. on Functional Programming Languages and Computer Architecture*, 1981, 33–39.

[15] Paul Broome, "Transformation of Parallel Programs with Higher Order Relational Operators ," University of Delaware, Technical Report No. 86-02, 1985.

[16] Philip Wadler, "Deforestation: Transforming programs to eliminate trees," in *LNCS*, H. Ganzinger, ed. #300, Springer-Verlag, Berlin, 1988, 344–358.

No of Copies	Organization
(Unclass., unlimited) 12 (Unclass., limited) 2 (Classified) 2	Administrator Defense Technical Info Center ATTN: DTIC-DDA Cameron Station Alexandria, VA 22304-6145
1	HQDA (SARD-TR) WASH, DC 20310-0001
1	Commander US Army Materiel Command ATTN: AMCDRA-ST 5001 Eisenhower Avenue Alexandria, VA 22333-0001
1	Commander US Army Laboratory Command ATTN: AMSLC-DL Adelphi, MD 20783-1145
2	Commander Armament RD&E Center US Army AMCCOM ATTN: SMCAR-MSI Picatinny Arsenal, NJ 07806-5000
2	Commander Armament RD&E Center US Army AMCCOM ATTN: SMCAR-TDC Picatinny Arsenal, NJ 07806-5000
1	Director Benet Weapons Laboratory Armament RD&E Center US Army AMCCOM ATTN: SMCAR-LCB-TL Watervliet, NY 12189-4050
1	Commander US Army Armament, Munitions and Chemical Command ATTN: SMCAR-ESP-L Rock Island, IL 61299-5000
1	Commander US Army Aviation Systems Command ATTN: AMSAV-DACL 4300 Goodfellow Blvd. St. Louis, MO 63120-1798
1	Director US Army Aviation Research and Technology Activity Ames Research Center Moffett Field, CA 94035-1099

No of Copies	Organization
1	Commander US Army Missile Command ATTN: AMSMI-RD-CS-R (DOC) Redstone Arsenal, AL 35898-5010
1	Commander US Army Tank Automotive Command ATTN: AMSTA-TSL (Technical Library) Warren, MI 48397-5000
1	Director US Army TRADOC Analysis Command ATTN: ATAA-SL White Sands Missile Range, NM 88002-5502
(Class. only) 1	Commandant US Army Infantry School ATTN: ATSH-CD (Security Mgr.) Fort Benning, GA 31905-5660
(Unclass. only) 1	Commandant US Army Infantry School ATTN: ATSH-CD-CSO-OR Fort Benning, GA 31905-5660
1	The Rand Corporation P.O. Box 2138 Santa Monica, CA 90401-2138
(Class. only) 1	Air Force Armament Laboratory ATTN: AFATL/DLODL Eglin AFB, FL 32542-5000
	<u>Aberdeen Proving Ground</u> Dir, USAMSAA ATTN: AMXSY-D AMXSY-MP, H. Cohen Cdr, USATECOM ATTN: AMSTE-TO-F Cdr, CRDEC, AMCCOM ATTN: SMCCR-RSP-A SMCCR-MU SMCCR-MSI

No. Copies	*Organization*
2	Director US Army Research Office ATTN:Dr. William Sander Dr. Jerry Andersen PO Box 12211 Research Triangle Park, NC 27709-2211
1	Director US Army Tank Automotive Command ATTN:Dr. William Jackson AMSTA-RSA Warren, MI 48397-5000
1	Dr. Stephen Wolff, Director Division of Networking and Communications National Science Foundation 1800 G St, NW Washington, DC 20550
1	Prof. Anil Nerode Mathematical Sciences Institute Cornell University 294 Caldwell Hall Ithaca, NY 14853-2602
1	Prof. Roger D. Maddux Department of Mathematics 400 Carver Hall Iowa State University Ames, Iowa 50011

No. Copies	*Organization*
1	Prof. Dale Miller Computer and Information Sciences University of Pennsylvania Philadelphia, PA 19104-6389
1	Dr. Sanjai Narain Rand Corporation 1700 Main Street Santa Monica, CA 90406

Aberdeen Proving Ground

Director, BRL
ATTN:SLCBR-SE
Mr. Morton A. Hirschberg
Mr. Richard Kaste
Dr. A. Brinton Cooper, III
Mr. Michael John Muuss
Mr. Andrew Thompson
Mr. Edwin O. Davisson

DISTRIBUTION LIST

1	Dr. David Chan Hewlett-Pachard Laboratories Stoke Gifford, Bristol, England
1	Prof. Philip Wadler Department of Computing Science University of Glasgow Glasgow G12 8QQ, Scotland

USER EVALUATION SHEET/CHANGE OF ADDRESS

This laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers below will aid us in our efforts.

1. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) ____

2. How, specifically, is the report being used? (Information source, design data, procedure, source of ideas, etc.) ____

3. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. ____

4. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) ____

BRL Report Number ________________ Division Symbol ________________

Check here if desire to be removed from distribution list. ____

Check here for address change. ____

Current address: Organization ________________________
Address ________________________
____

-----------------------------FOLD AND TAPE CLOSED-----------------------------

Director
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T(NEI)
Aberdeen Proving Ground, MD 21005-5066

OFFICIAL BUSINESS

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY LABEL
FIRST CLASS PERMIT NO 12062 WASHINGTON D C
POSTAGE WILL BE PAID BY DEPARTMENT OF THE ARMY

Director
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T(NEI)
Aberdeen Proving Ground, MD 21005-9989